INFORME

TRABAJO PRÁCTICO 3

ARQUITECTURA DE COMPUTADORAS - AÑO 2023

TEMA: Diseño de pipeline segmentado

DOCENTE

Rodriguez Santiago

Pereyra Martin

ALUMNOS	CARRERA	MATRÍCULA
Almarcha C., Matias A.	Ing. en Computación	41349330
De la Cruz, Francisco A.	Ing. en Computación	41481173

Índice temático

1 - Introducción

La arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages) es una familia de microprocesadores diseñada con un tipo de arquitectura RISC.

Esta arquitectura tiene en mente los siguientes principios

- Instruction set simple de carga y almacenamiento (Load/Store)
- Eficiencia del pipeline segmentado.
- Eficiente para la generación automática de código (compilación)

Hay múltiples instruction sets, conocidos como MIPS I, MIPS II, MIPS III, MIPS IV y MIPS 32/64

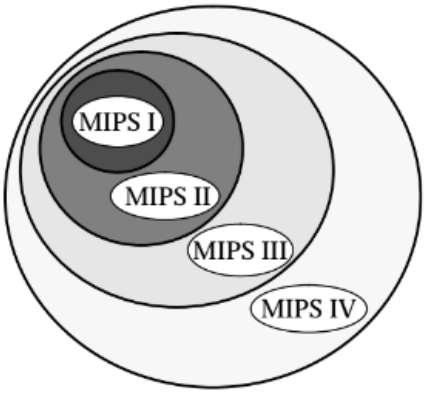


Fig 01: Extensiones a la arquitectura MIPS

2 - Objetivos

Se debe implementar el procesador MIPS segmentado en las siguientes etapas:

- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Write back

Las instrucciones a implementar son un subconjunto del set de instrucciones del MIPS:

- R-type:
  - SLL, SRL, SRA, SLLV, SRLV, SRAV, ADDU, SUBU, AND,OR, XOR, NOR, SLT.
- I-Type:
  - LB, LH, LW, LWU, LBU, LHU, SB, SH, SW, ADDI, ANDI, ORI, XORI, LUI, SLTI, BEQ, BNE, J, JAL.
- J-Type:
  - JR, JALR.

Los códigos de operación (OPCODE) y los códigos de función (FUNCTION) se obtuvieron de la documentación oficial MIPS IV: Instruction Set

La pipeline debe implementar detección y control de riesgos. Se debe simular una unidad de debug que envíe información hacia y desde el procesador mediante UART.
La información es la siguientes:

- Contador de programa (PC).
- Contenido de los 32 registros.
- Contenido de la memoria de datos usada.
- Cantidad de ciclos de reloj ejecutados

Una vez cargado el programa a ejecutar, el procesador debe permitir dos modos de operación:

- **Continuo**: se envía un comando a la FPGA por la UART y se inicia la ejecución del programa hasta llegar al final del mismo (instrucción HALT). Luego, se muestra la información pedida en la pantalla.
- **Paso a paso**: se envía un comando a la FPGA por la UART, se ejecuta un ciclo de clock y se muestra la información pedida. Seguido de recibir nuevamente un comando ya sea un ciclo más o en modo continuo hasta que termine de ejecutarse todo el programa.

El clock del sistema debe crearse utilizando el ip-core correspondiente.

Se debe mostrar el reporte de timing con la frecuencia máxima de reloj soportada.

Se debe incluir un ensamblador para convertir el lenguaje Assembly a código de máquina y se debe trasmitir mediante UART.

Se debe validar el desarrollo mediante **Test Bench**.

Instrucciones

MIPS posee tres tipos de instrucciones R, I y J.

Descripción de cada instrucción

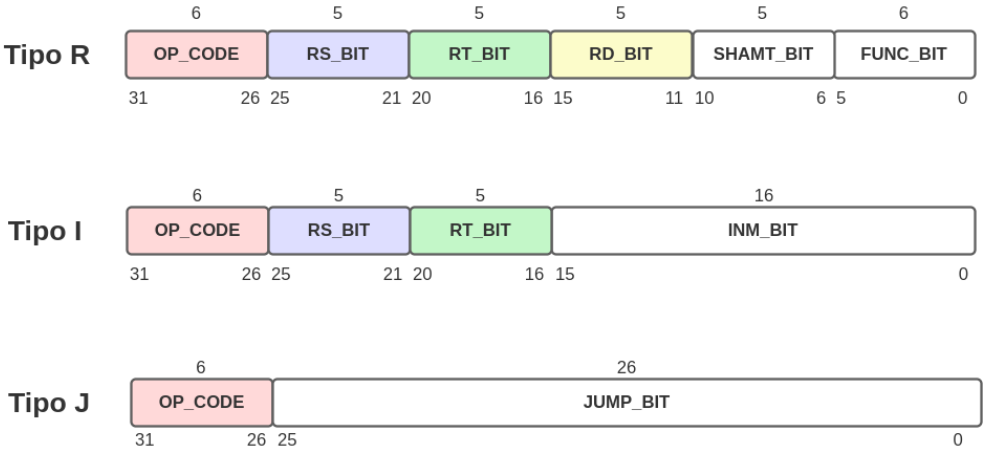


Fig 02: Tipos de instrucciones MIPS

Se implementa un subconjunto del ISA MIPS

Los códigos de operación (OPCODE) y los códigos de función (FUNCTION) se obtuvieron de la documentación oficial [MIPS IV: Instruction Set]

R-Type		
OPERATION	**OPCODE**	**FUNCTION**
AND	000000	100100
OR	000000	100101
XOR	000000	100110
NOR	000000	100111
ADDU	000000	100001
SUBU	000000	100011
SLT	000000	101010
SRL	000000	000010
SLL	000000	000000
SRA	000000	000011
SLLV	000000	000100
SRLV	000000	000110
SRAV	000000	000111

I-Type	
OPERATION	**OPCODE**
LB	100000
LH	100001
LW	100011
LWU	100111
LBU	100100
LHU	100101
SB	101000
SH	101001
SW	101011
ADDI	001000
ANDI	001100
ORI	001101
XORI	001110
LUI	001111
SLTI	001010
BEQ	000100
BNE	000101
J	000010
JAL	000011

J-Type		
OPERATION	**OPCODE**	**FUNCTION**
JR	000000	001000
JALR	000000	001001

3 - Desarrollo

3.1 - Módulos base

Compuestos por los componentes básicos para construir las etapas de la pipeline

ALU

El submódulo alu (Unidad de Lógica Aritmética) es un componente fundamental utilizado en la etapa de ejecución de un procesador para llevar a cabo operaciones aritméticas y lógicas en datos. Este submódulo toma dos operadores, realiza una operación basada en una señal de control y produce un resultado.

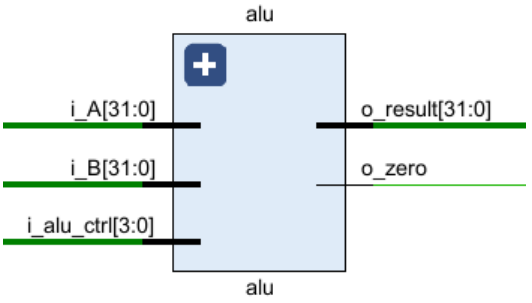


Fig 03: ALU

OPERACIÓN	**OPCODE**
SLL	0000
SRL	0001
SRA	0010
ADD	0011
SUB	0100
AND	0101
OR	0110
XOR	0111

NOR	1000
SLT	1001
LUI	1010

ALU Control

Este módulo de control de ALU (Unidad Lógico-Aritmética) en un procesador MIPS (Microprocessor without Interlocked Pipeline Stages) se encarga de generar señales de control que indican a la ALU qué operación debe realizar, según el tipo de instrucción que se esté ejecutando.

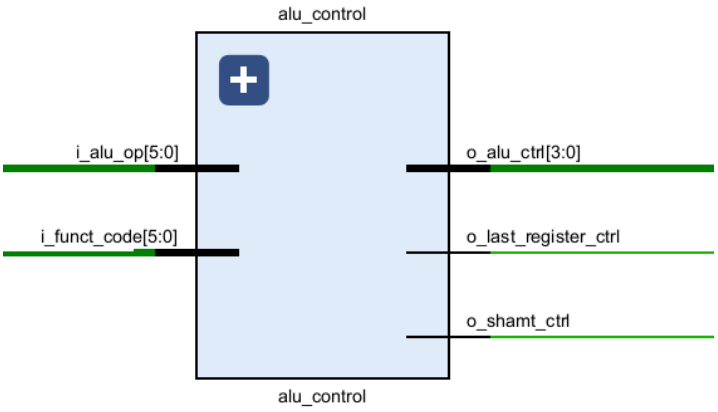


Fig 04: Control de la ALU

Banco de registros

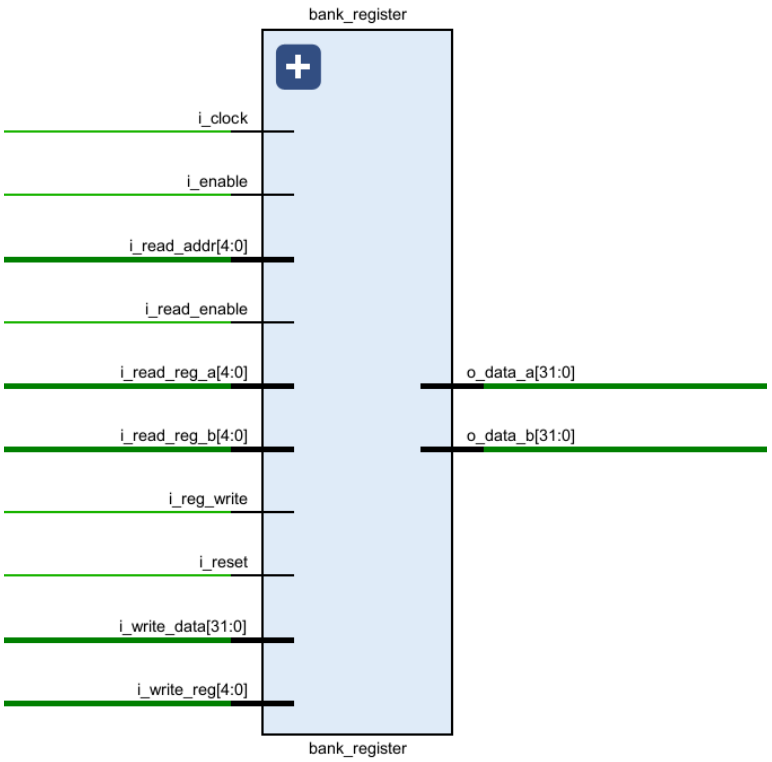


Fig 05: Banco de Registros

Un banco de registros es una colección de registros donde cualquier registro puede leerse o escribirse especificando su número.

Los registros de 32 bits del procesador se agrupan en una estructura denominada banco de registros, de él se leen los operandos de cada instrucción en la etapa de Instruction Decode.

Estos son 32 registros que se numeran del 0 al 31 y el propósito de cada uno se encuentra detallado en la **Tabla 02**.

Tabla 02: Registros del CPU

Number	**Name**	**Comments**
R0	$zero, $r0	Always zero
R1	$at	Reserved for assembler

R2, R3	$v0, $v1	First and second return values, respectively
R4, ..., R7	$a0, ..., $a3	First four arguments to functions
R8, ..., R15	$t0, ..., $t7	Temporary registers
R16, ..., R23	$s0, ..., $s7	Saved registers
R24, R25	$t8, $t9	More temporary registers
R26, R27	$k0, $k1	Reserved for kernel (operating system)
R28	$gp	Global pointer
R29	$sp	Stack pointer
R30	$fp	Frame pointer
R31	$ra	Return address

Memoria de datos

La memoria de instrucciones se utiliza para almacenar el programa o conjunto de instrucciones que el procesador debe ejecutar. La CPU accede a la memoria de instrucciones para buscar las instrucciones del programa, las carga en su caché de instrucciones y las decodifica para que puedan ser ejecutadas.

Por otro lado, la memoria de datos se utiliza para almacenar los datos que el programa necesita para ejecutarse. Los datos pueden ser entradas del usuario, resultados intermedios o finales, entre otros. La CPU accede a la memoria de datos para leer o escribir en ella.

La separación de la memoria de instrucciones y de datos permite una organización más eficiente de la memoria y reduce la complejidad del procesador, lo que se traduce en una mejora en el rendimiento.

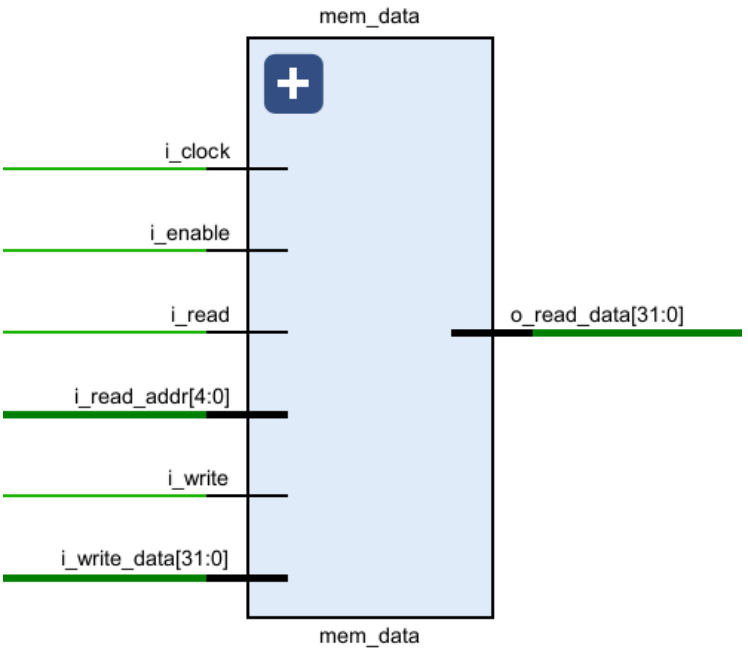


Fig 06: Memoria de Datos

Memoria de instrucciones

La memoria de instrucciones en este módulo almacena las operaciones del programa. Puede recuperar instrucciones desde ubicaciones de memoria específicas cuando se activa la lectura. Asimismo, permite cargar nuevas instrucciones en la memoria cuando se habilita la escritura en direcciones específicas. Cada instrucción tiene un ancho de 32 bits y se accede mediante un bus de direcciones de lectura de 10 bits.

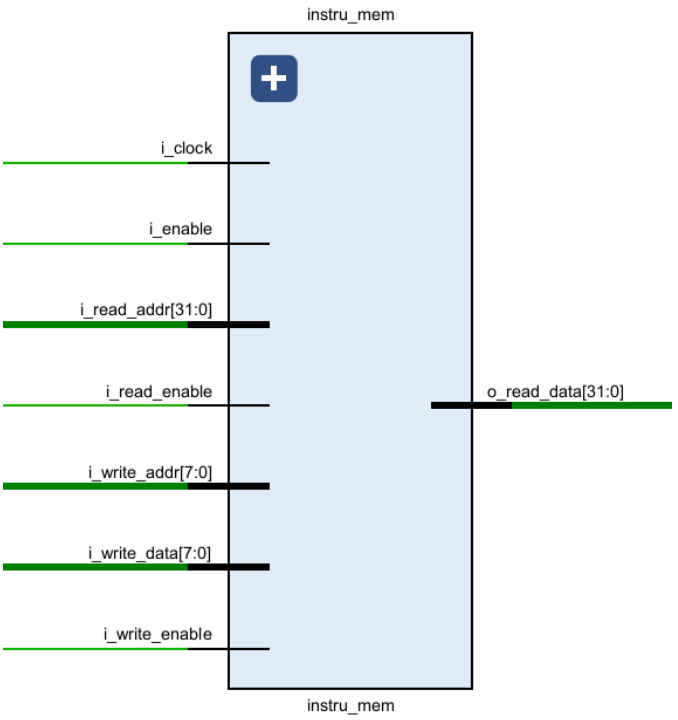


Fig 07: Memoria de Instrucciones

Memory Controller

Maneja la lectura y escritura de datos con diferentes tamaños de palabra y considera si los datos son signados o no signados.

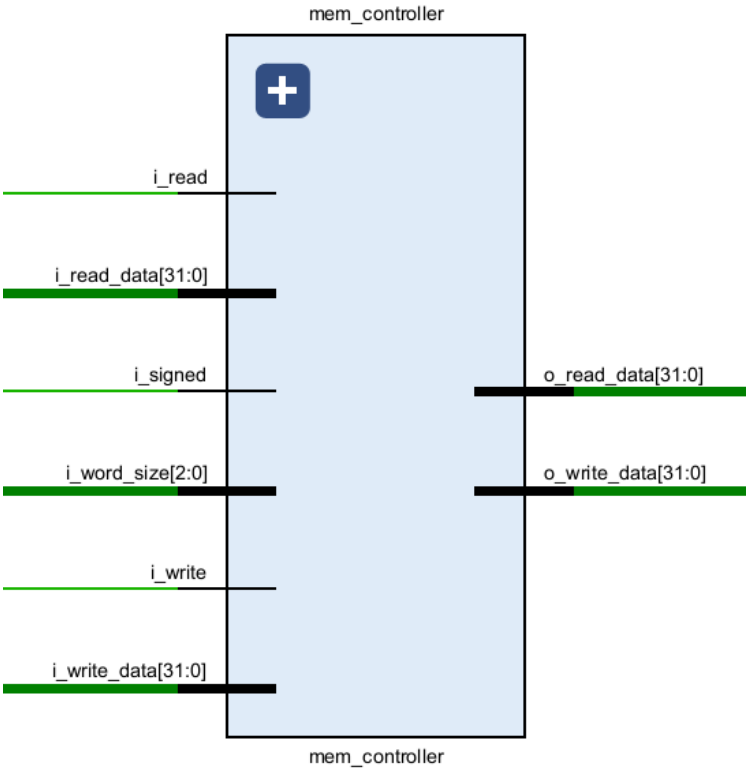


Fig 08: Controlador de Memoria

Program Counter

El Program Counter se incrementa automáticamente después de cada instrucción ejecutada, de modo que apunte a la siguiente instrucción.

Es muy importante en la ejecución de un programa, ya que controla la secuencia de instrucciones a ejecutar. Sin él, el procesador no sabría en qué dirección de memoria buscar la próxima instrucción a ejecutar, lo que impediría la ejecución de un programa.

El valor del Program Counter se puede modificar mediante *saltos o saltos condicionales*, que permiten al procesador saltar a otra dirección de memoria para ejecutar una instrucción diferente. También se puede modificar el valor del Program Counter mediante llamadas y retornos de subrutinas, que permiten al procesador saltar a una dirección de memoria diferente y luego volver a la instrucción siguiente a la llamada a la subrutina.

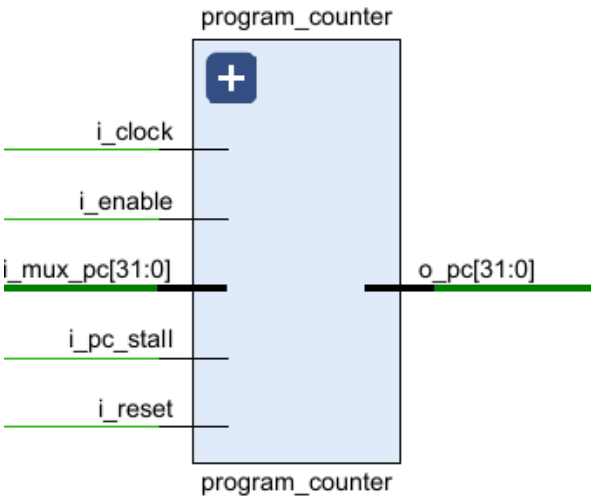


Fig 09: Contador de Programa

Unit Control

El submódulo se encarga de generar señales de control en función del tipo de instrucción. Recibe el código de operación y el código de función como entradas y genera señales de control como:

- **o_alu_op**: Señal que configura la operación que realizará la ALU (por ejemplo, suma, resta, AND, OR).
- **o_signed**: Indica si se requiere una operación con signo.
- **o_reg_dest**: Controla si se debe escribir en un registro de destino.
- Otras señales relacionadas con la lectura y escritura de registros, acceso a memoria y operaciones de salto y halts.

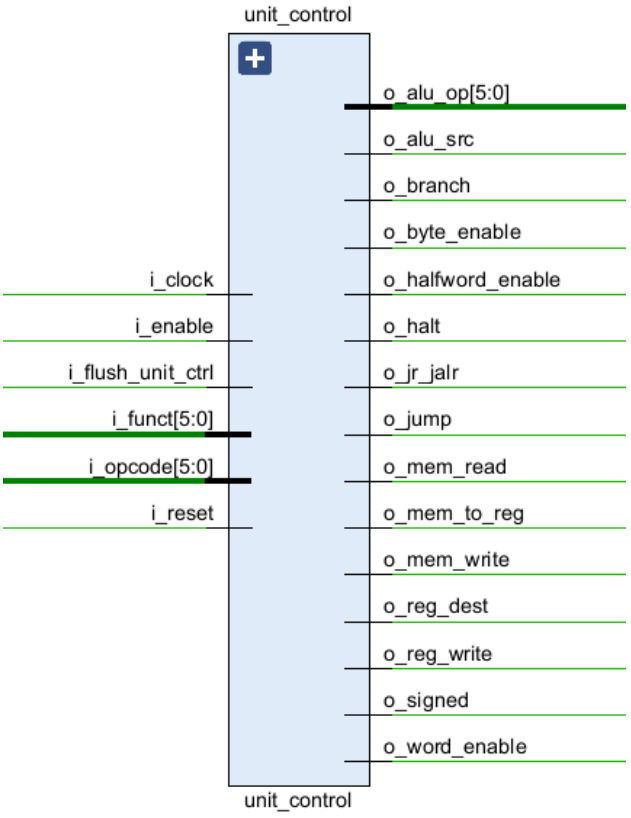


Fig 10: Unidad de Control

Forward Unit

Este submódulo representa una "forwarding unit" diseñada para un procesador. El propósito de esta unidad es *gestionar* el reenvío (forwarding) de datos entre diferentes etapas del pipeline del procesador para evitar problemas de dependencias de datos y mejorar el rendimiento.

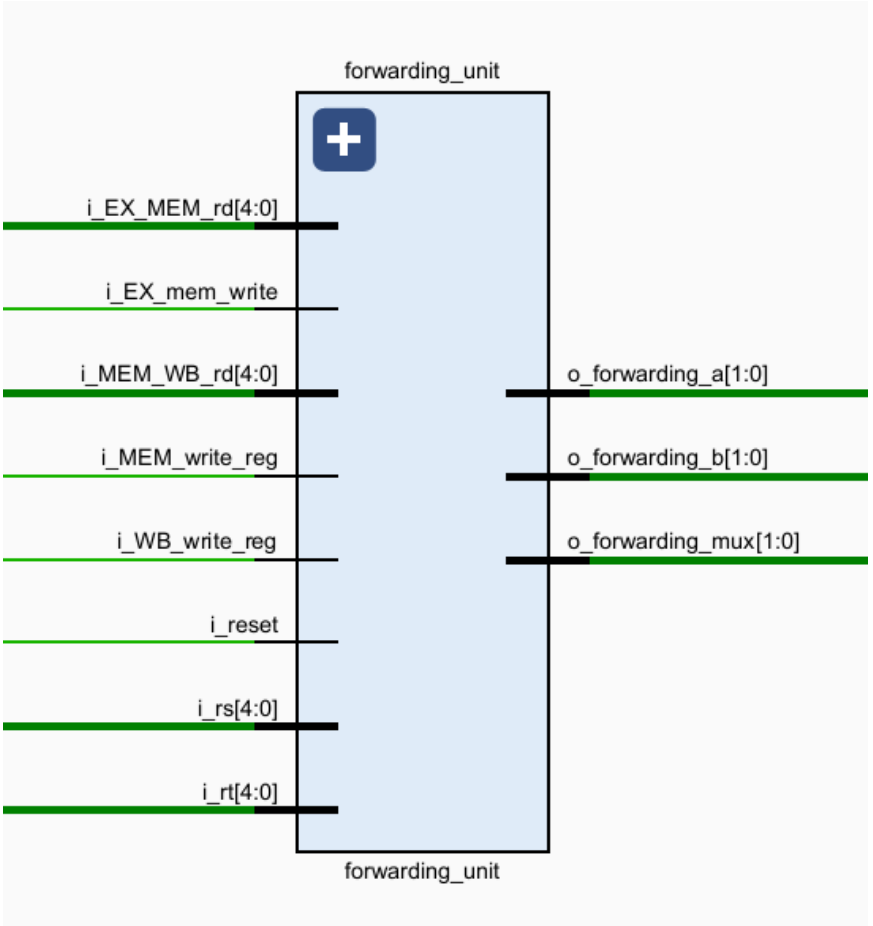


Fig 11: Unidad de Cortocircuito

Unit Stall

Su propósito principal es gestionar el flujo de control y los posibles stalls en un pipeline de ejecución de instrucciones de un procesador. El módulo toma varias entradas de control y datos, y produce salidas que afectan diferentes etapas del pipeline.

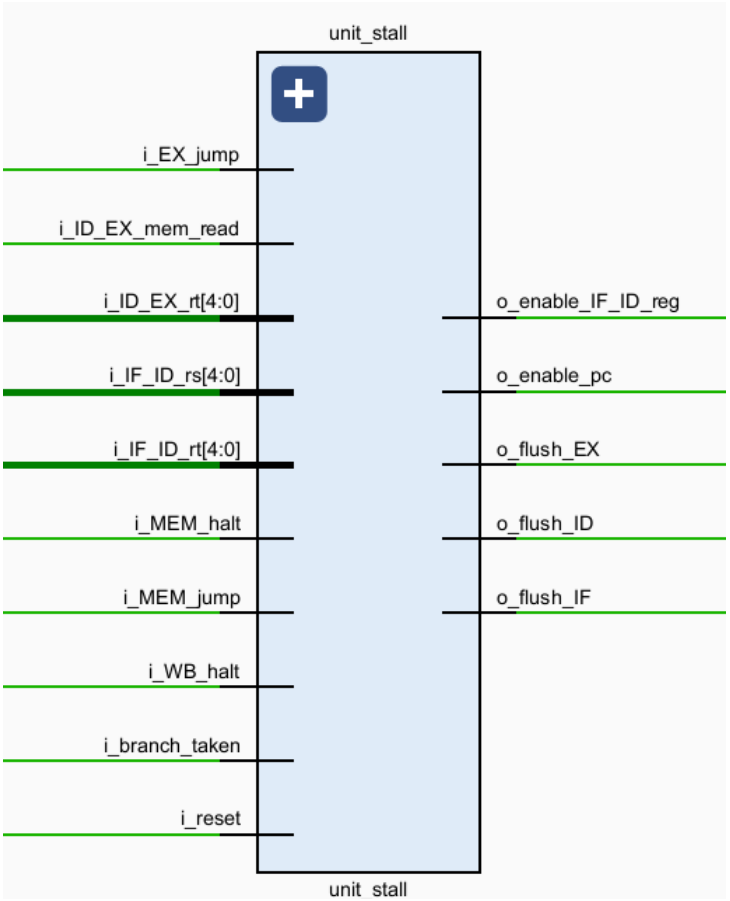


Fig 12: Unidad de Stall

3.2 - Etapas

Instruction Fetch

Esta etapa es la encargada de leer las instrucciones de la memoria de instrucciones. Para ello utiliza la dirección almacenada en el Contador del Programa (PC).

La dirección se incrementa en 4 y se escribe de nuevo en el PC para prepararse para el siguiente ciclo de reloj. En caso de haber un branch, la señal PC_SRC cambia el selector del multiplexor que indica la siguiente dirección de memoria.

Tanto la instrucción como la dirección del PC se almacenan en el registro de IF/ID por si alguna instrucción, por ejemplo BEQ, la necesita con posterioridad.

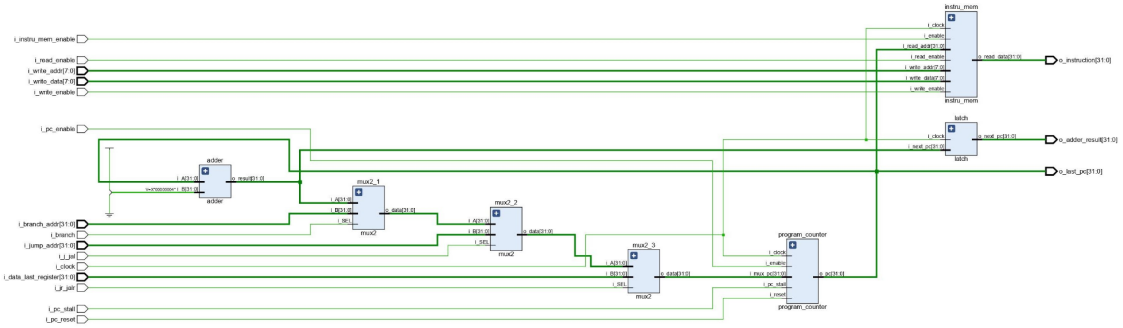


Fig 13: Etapa Instruction Fetch

Decode

Esta etapa es la encargada de leer las instrucciones provistas por la etapa de Instruction Fetch y decodificar lo que la función debe hacer obteniendo operandos y señales de control. Utiliza el banco de registros junto a unidades funcionales que proporcionan el control de flujo y de datos del procesador.

Estas unidades son:

- **Unidad de Control**: Controla el flujo del programa y la ejecución de las operaciones en cada etapa.
- **Banco de registros**
- **Módulo de concatenación**
- **Módulo de extensión de signo**

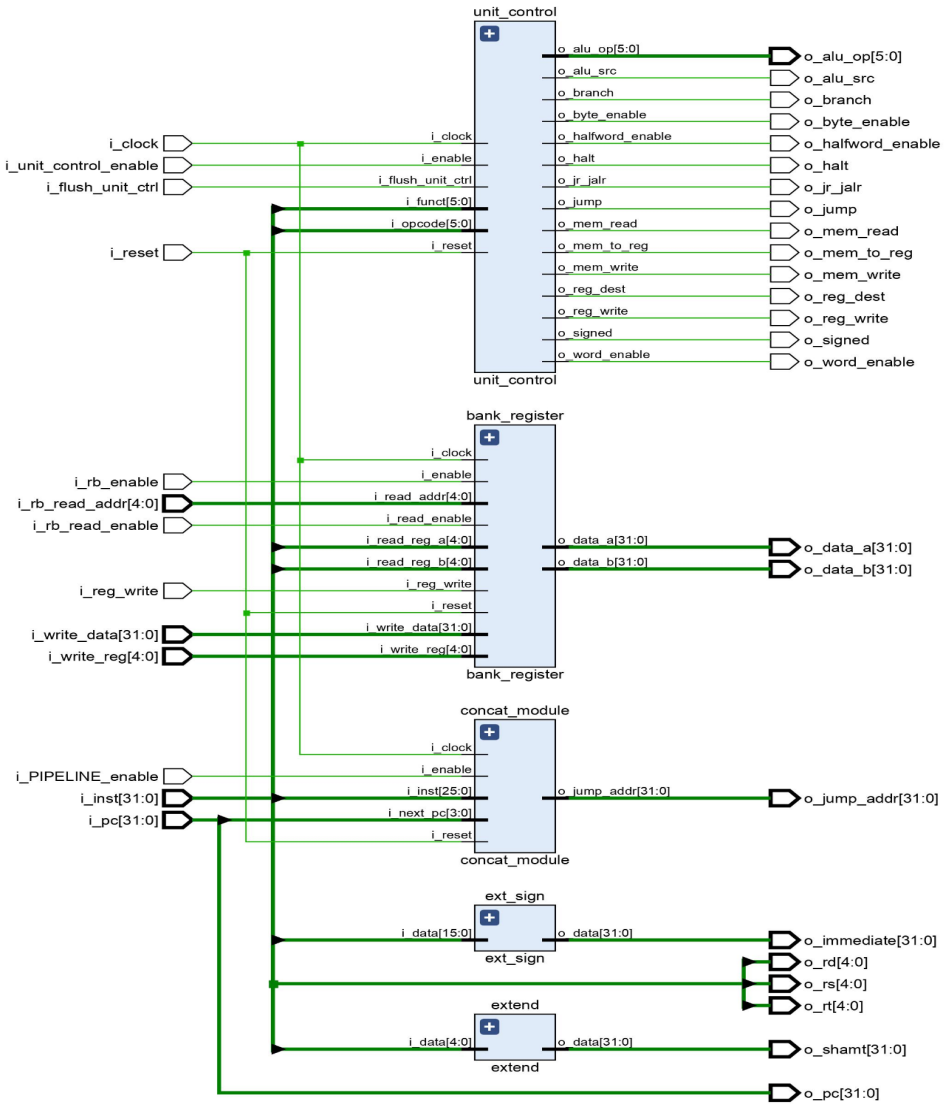


Fig 14: Etapa Instruction Decode

Execution

La etapa de ejecución del procesador es gestionada por el módulo EXECUTE, que se encarga de realizar operaciones aritméticas, lógicas y de control de flujo necesarias para ejecutar una instrucción dada. Es una parte crucial del procesador, encargada de ejecutar instrucciones y determinar los resultados de las operaciones. Durante esta etapa, se interactúa con otros módulos del procesador, como el módulo de memoria (MEM) y el módulo de escritura en registros (WB), para completar la ejecución de instrucciones.

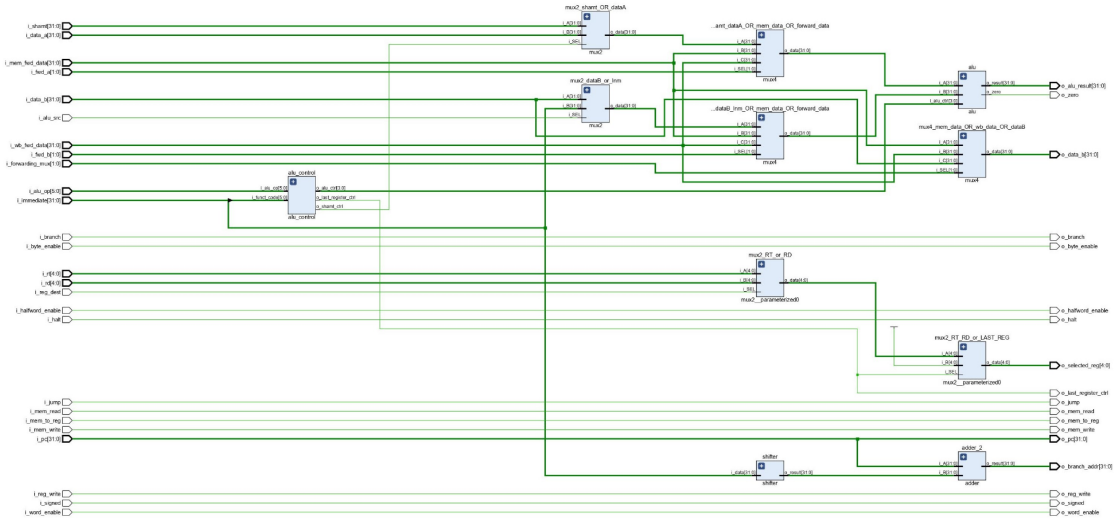


Fig 15: Etapa de Ejecucion

Memory

El módulo MEMORY función principal es permitir la escritura y lectura de datos en la memoria, además de proporcionar salidas y controlar el flujo de datos.

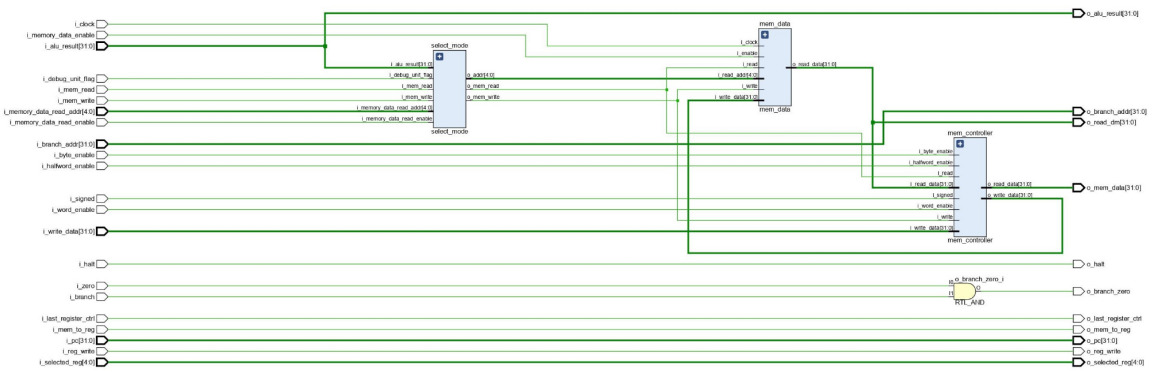


Fig 16: Etapa Memoria

Write back

El módulo *WRITE BACK* es una parte esencial de la etapa de escritura de un pipeline en un procesador. Su función principal es determinar si se realizará una escritura en los registros del procesador y seleccionar los datos que se escribirán.

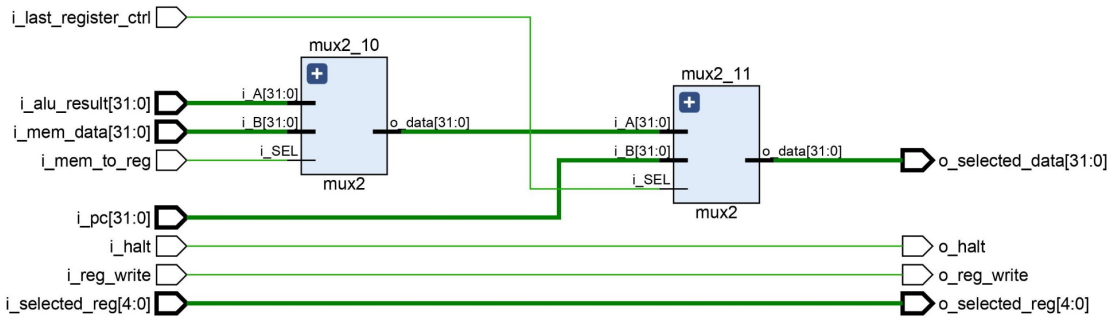


Fig 17: Etapa Write Back

Unidad de Debug

Diagrama de estados debug unit

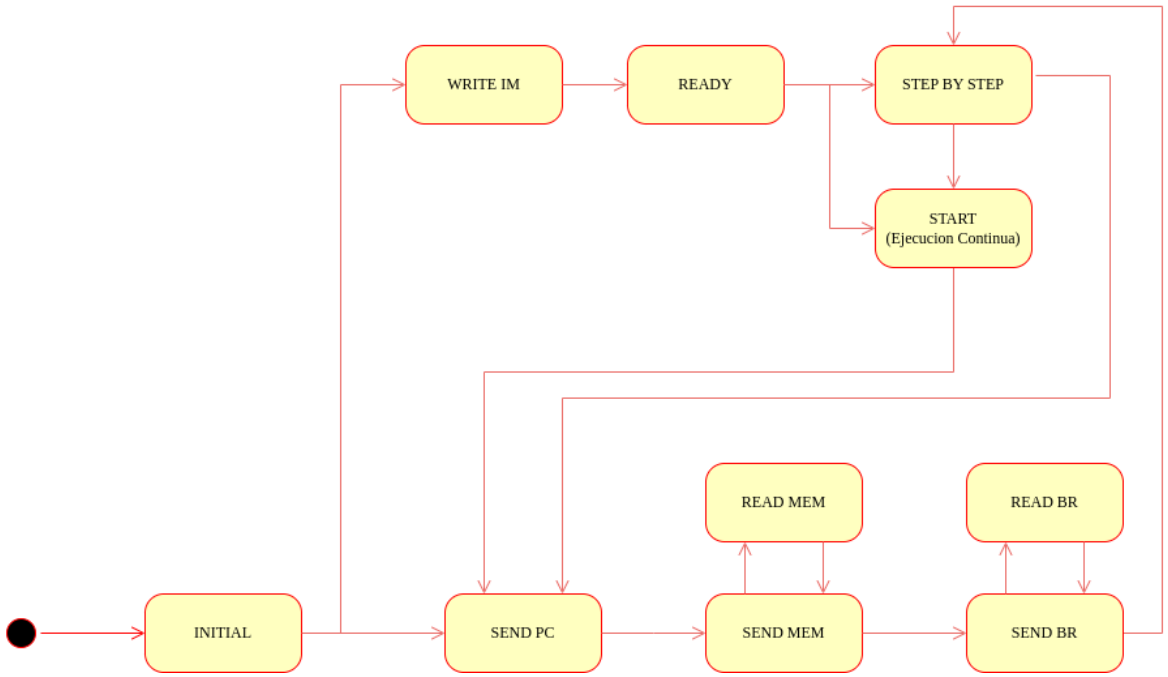


Fig 18: Máquina de estados de la Debug Unit

Los estados son los siguientes:

Estado	Descripción
INITIAL	Estado inicial donde la unidad de depuración espera comandos
READY	Listo para recibir comandos específicos, de ejecución paso a paso o ejecución continua.
START	Inicia la ejecución normal del sistema, habilitando operaciones en memoria y control general.
STEP BY STEP	Permite la ejecución paso a paso y puede enviar la dirección del contador de programa.
WRITE IM	Escribe datos en la memoria de instrucciones y vuelve al estado READY.
SEND PC	Envía la dirección del contador de programa a través de UART.

READ BR	Inicia la lectura del banco de registros.
SEND BR	Envía datos del banco de registros a través de UART.
READ MEM	Inicia la lectura de la memoria de datos.
SEND MEM	Envía datos de la memoria de datos a través de UART.

4 - Interfaz de Usuario

El script proporcionado es una interfaz gráfica de usuario (GUI) en Python utilizando el módulo Tkinter. La **GUI** está diseñada para interactuar con un sistema MIPS-DLX a través de una conexión UART (Universal Asynchronous Receiver/Transmitter). El script proporciona funcionalidades como seleccionar un puerto UART, cargar un archivo de instrucciones, compilar el código assembly, enviar el programa al sistema y ejecutar instrucciones paso a paso.

Requisitos previos

Antes de ejecutar el script, asegúrate de tener instaladas las siguientes bibliotecas utilizando:

```
python3 -m pip install tkinter
python3 -m pip install pyserial
```

Descripción de la interfaz

Selección de Puerto y Baudrate:

La ventana principal muestra un menú desplegable para seleccionar el puerto UART y otro para seleccionar la velocidad de baudios (baudrate).

Ventana de Selección de Archivo:

Al seleccionar un puerto, se abrirá una nueva ventana que te permitirá seleccionar un archivo de instrucciones en lenguaje assembly.

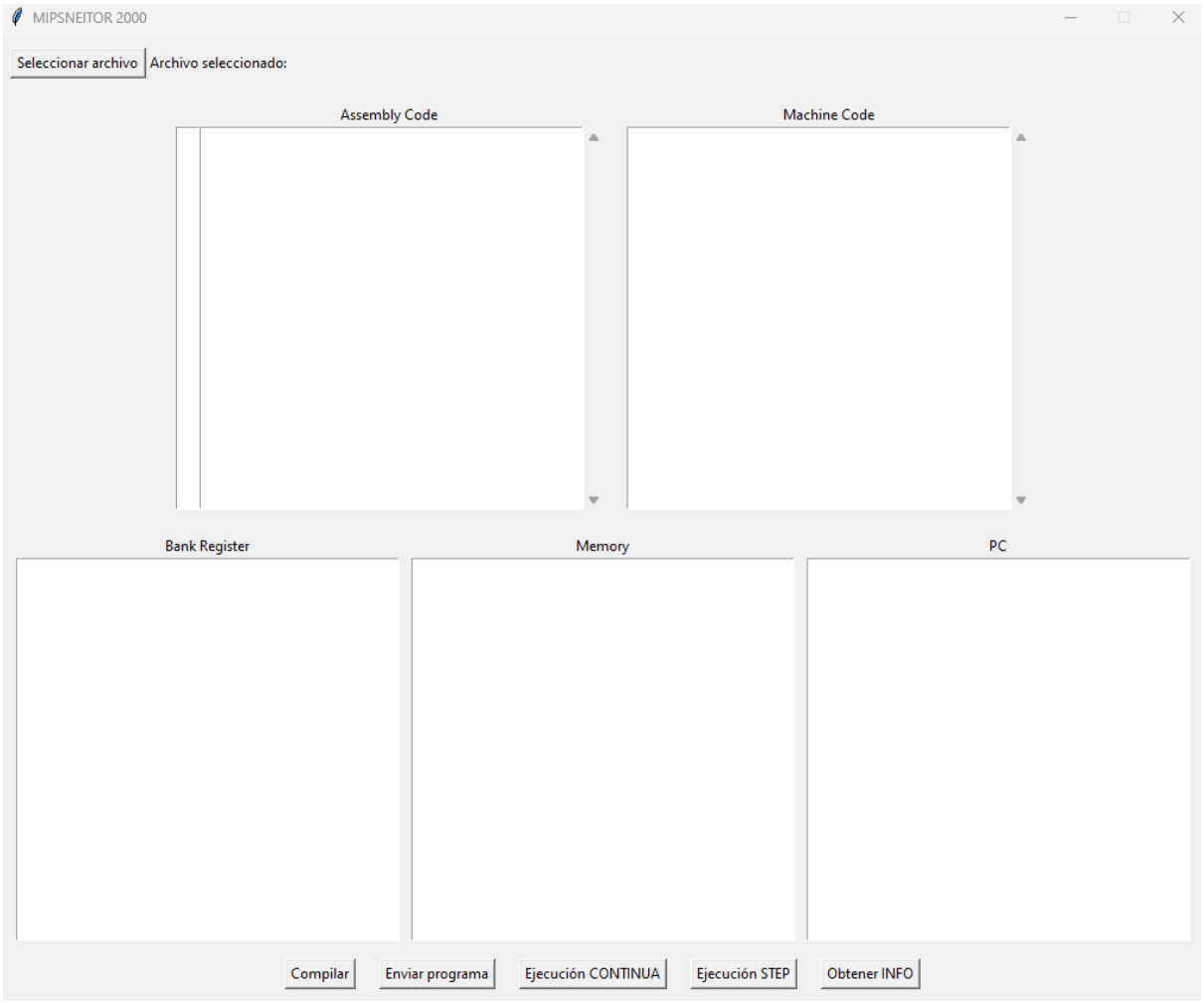
Editor de Código:

En la ventana de selección de archivo, verás dos áreas de texto. El área de la izquierda muestra el código assembly, y el área de la derecha mostrará el código compilado.

Botones de Acción:

En la parte inferior de la ventana de selección de archivo, hay varios botones de acción:

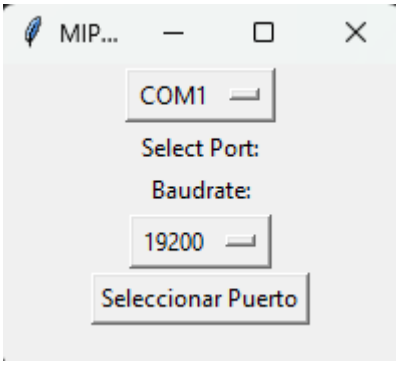
- **Compilar**: Compila el código assembly seleccionado.
- **Enviar Programa**: Envía el programa compilado al sistema.
- **Ejecución CONTINUA**: Inicia la ejecución continua del programa.
- **Ejecución STEP**: Ejecuta el programa paso a paso.
- **Obtener INFO**: Obtiene información del sistema.



Uso paso a paso

Seleccionar Puerto:

Abre la aplicación y selecciona el puerto UART y la velocidad de baudios. Haz clic en "Seleccionar Puerto".



Seleccionar Archivo de Instrucciones:

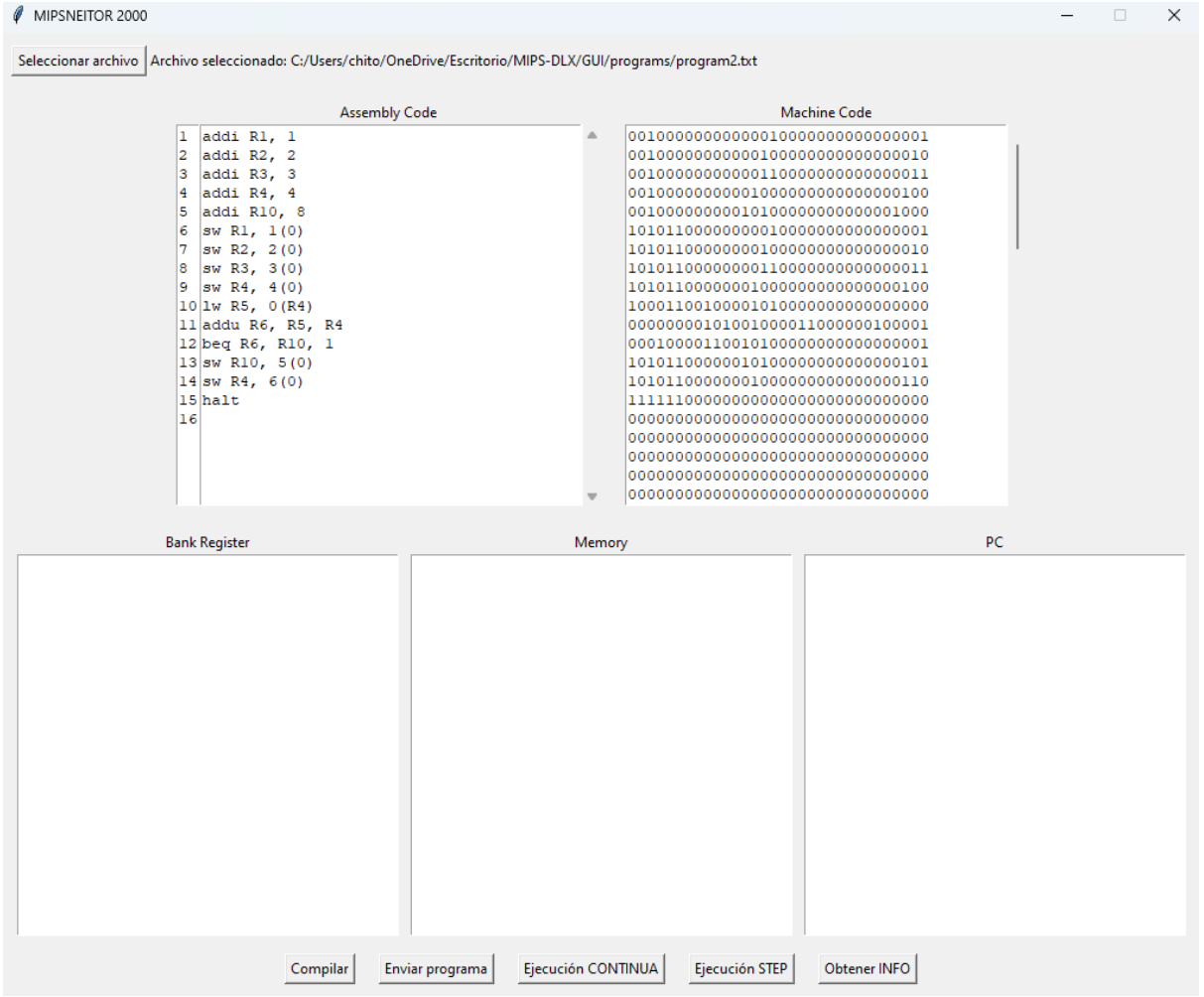
Después de seleccionar el puerto, la ventana se minimizará, y aparecerá una nueva ventana para seleccionar un archivo de instrucciones en lenguaje assembly. Selecciona el archivo deseado.

Compilar:

Haz clic en el botón "Compilar" para convertir el código assembly en instrucciones de máquina. El código compilado se mostrará en el área de texto de la derecha.

Enviar Programa:

Haz clic en "Enviar Programa" para enviar las instrucciones al sistema a través del puerto UART.



Ejecución Continua:

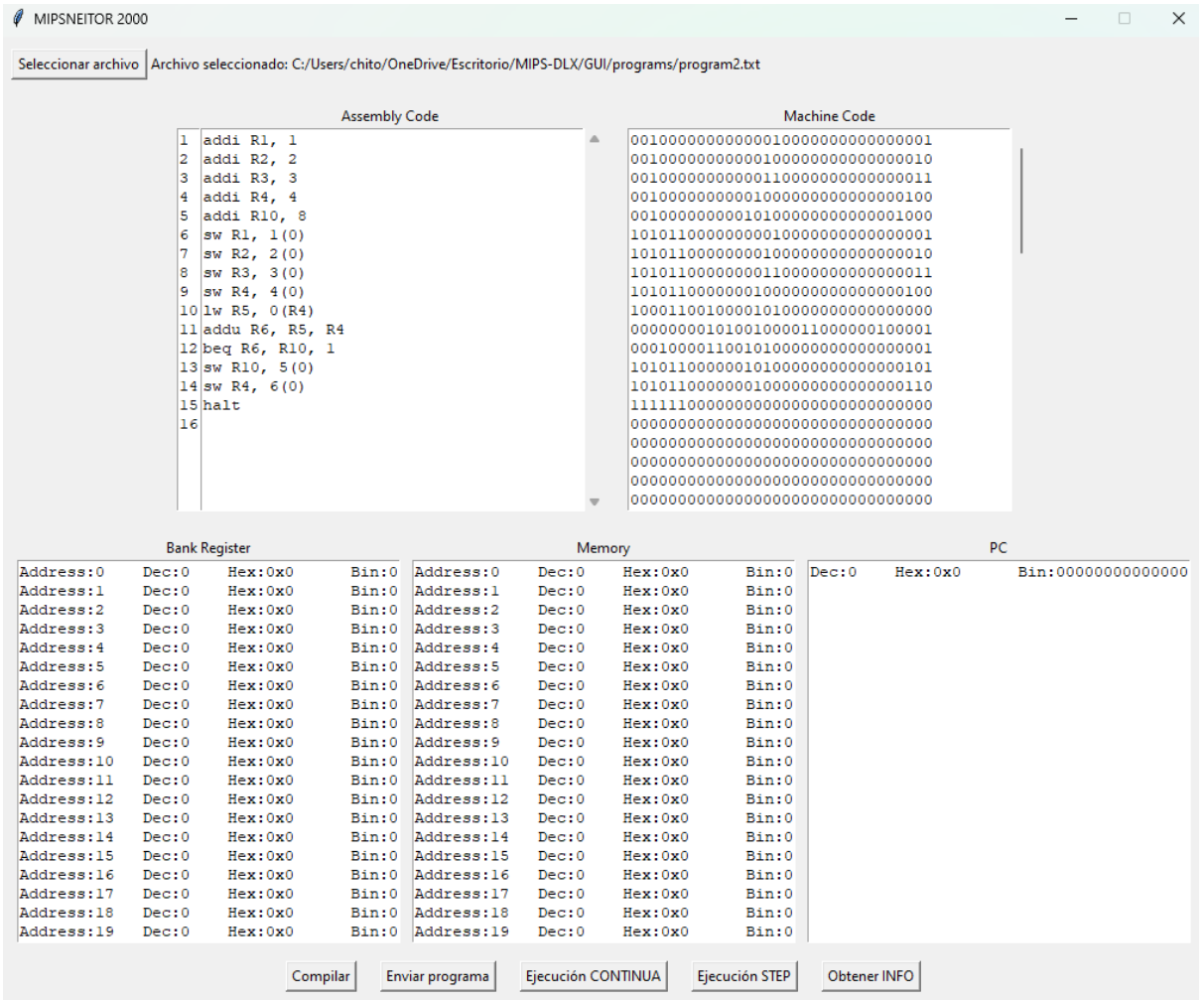
Haz clic en "Ejecución Continua" para ejecutar el programa de manera continua en el sistema.

Ejecución STEP:

Haz clic en "Ejecución Paso a Paso" para ejecutar el programa paso a paso en el sistema.

Obtener INFO:

Haz clic en "Obtener Información" para obtener información del sistema. Ten en cuenta que este botón sólo funcionará si no has enviado previamente el programa al sistema.

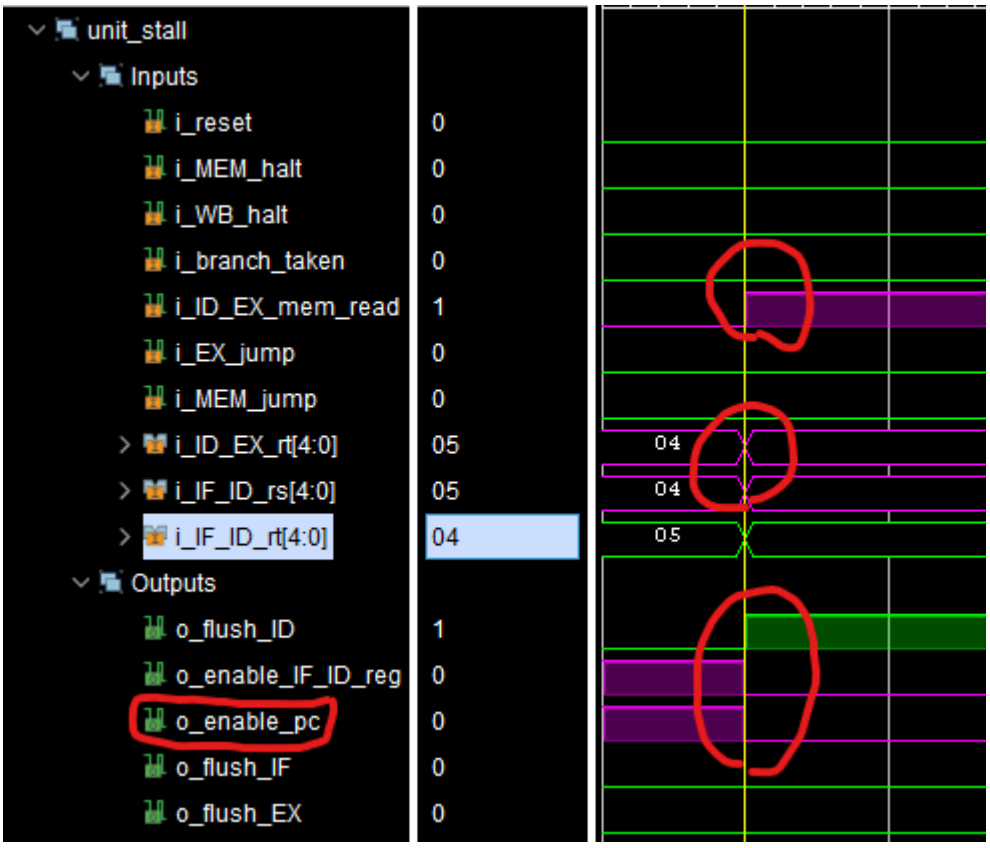


5 - Pipeline testing

Test Riegos Load

```
lw R5, 0(R4)
addu R6, R5, R4
```

```
if(((i_ID_EX_rt == i_IF_ID_rt) ||
    (i_ID_EX_rt == i_IF_ID_rs)) && i_ID_EX_mem_read) begin
    o_flush_IF              = 1'b0;
    o_flush_EX              = 1'b0;
    o_flush_ID              = 1'b1;
    o_enable_IF_ID_reg      = 1'b0;
    o_enable_pc             = 1'b0;     // disable PC
```



Test BEQ

Explicar codigo de abajo como funciona (se ve partes de codigo de diferentes etapas para mayor explicación

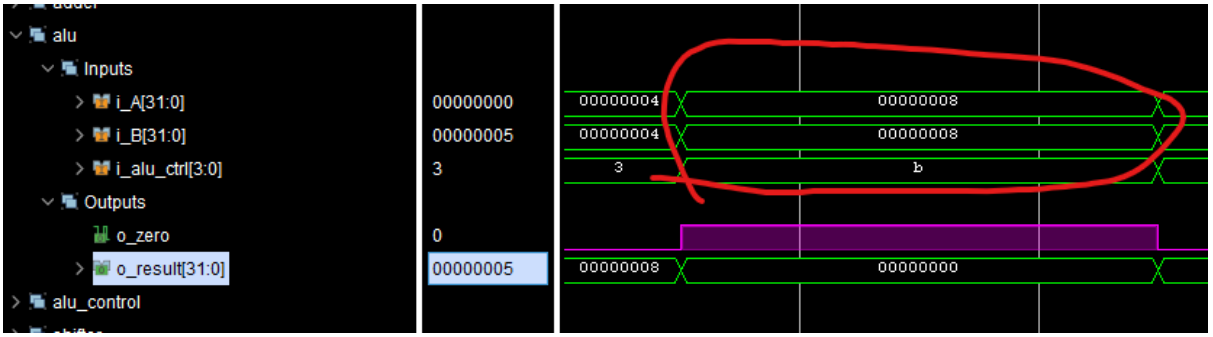
tener en cuenta que el salto se toma desde la etapa de memory porque recién ahí se envían las señales a la unit stall y la etapa de fetch. La unit stall es la encargada de limpiar o flushear las etapas porque al tomar el salto hay instrucciones en la pipe que al tomar el salto no tienen que afectar y la etapa fetch es la encargada de modificar el PC a la dirección del salto correspondiente.

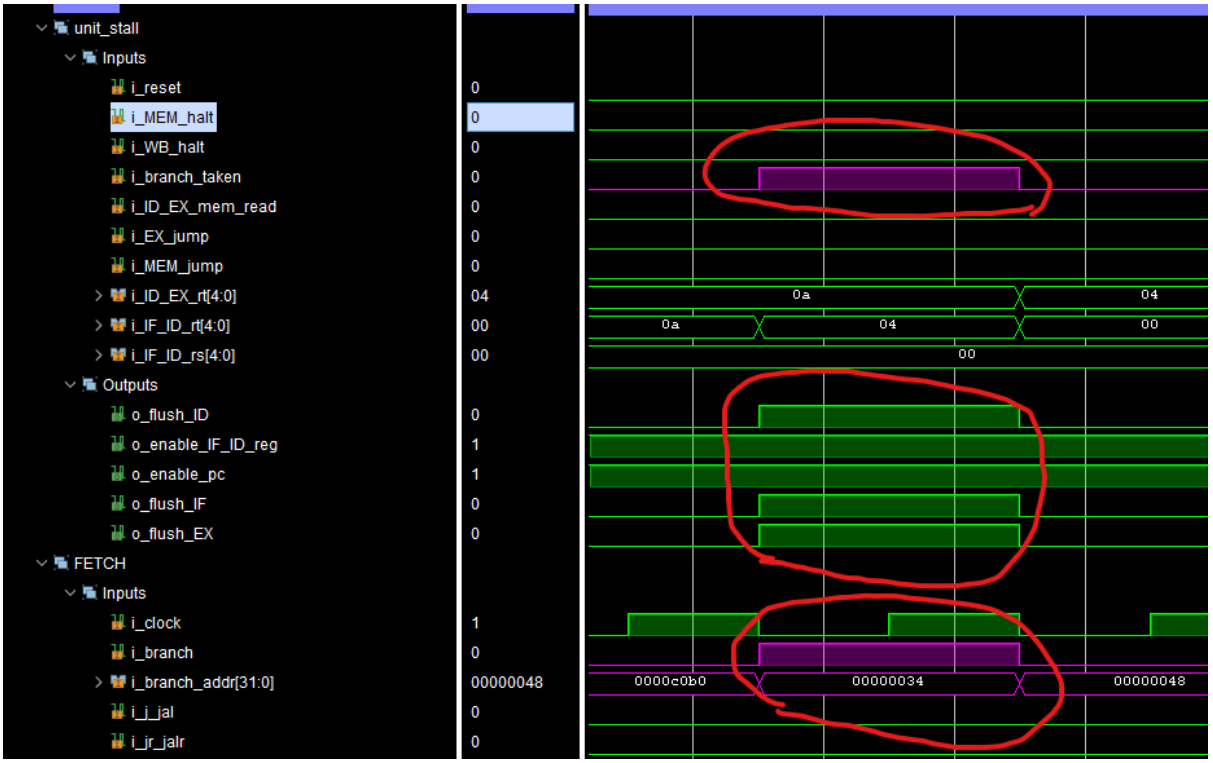
Código

```
addi R4, 4
addi R5, 4
addi R10, 8
addu R6, R5, R4
beq R6, R10, 1
```

```
// MEMORY send to FETCH
assign o_branch_zero = i_zero & i_branch;
assign o_branch_addr = i_branch_addr;
 // FETCH
  mux2 mux2_1
  (
      .i_SEL(i_branch),
      .i_A(adder_result),      // PC+1
      .i_B(i_branch_addr),     // branch
      .o_data(mux2_1_output)
  );
```

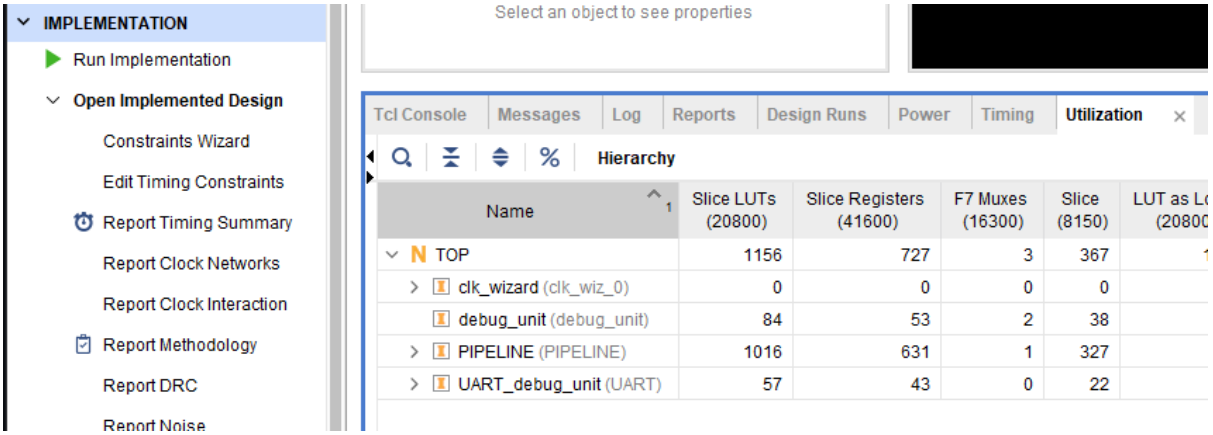
```
//Unit Stall
        i_branch_taken(o_branch_zero), // from MEM
        if(i_branch_taken) begin    // Hazards con branches
                //Flush
                o_flush_IF              = 1'b1;
                o_flush_EX              = 1'b1;
                o_flush_ID              = 1'b1; // DECODE
                o_enable_IF_ID_reg      = 1'b1;
                o_enable_pc             = 1'b1;
```



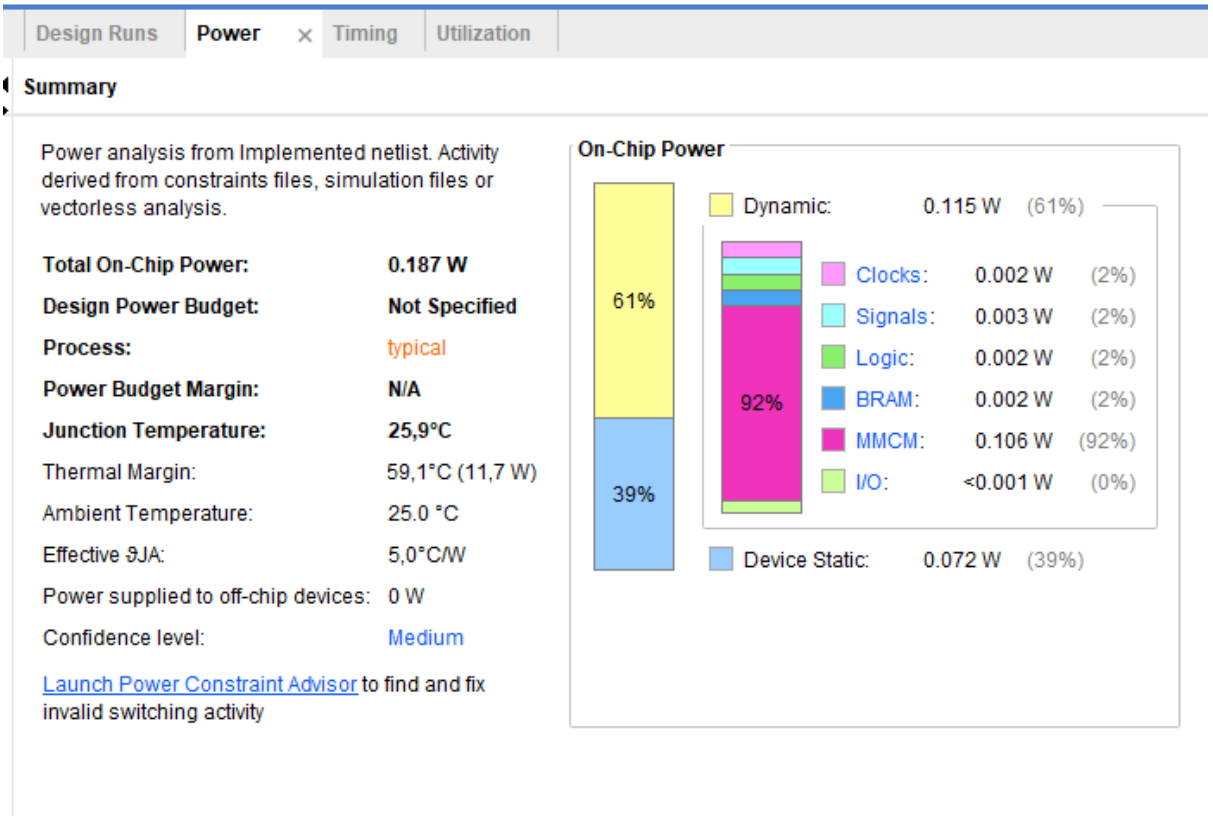


6 - Implementation 50MHz

Utilization Report



Power Report



Análisis de timing 50MHz

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	3,209 ns	Worst Hold Slack (WHS):	0,119 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2517	Total Number of Endpoints:	2517

All user specified timing constraints are met.

Setup:

Intervalo *antes* de la transición activa del clock durante el cual los datos deben mantenerse.

Hold:

Intervalo *luego* de la transición activa, ambos fueron tenidos en cuenta a la hora de analizar cómo se comportaba el procesador a diferentes frecuencias. Analizando los

Worst Negative Slack:

Cuando es **negativo**, significa que el delay del path combinacional es mayor que el período del clock. Por lo que se deberá disminuir la frecuencia del clock para así aumentar el periodo o bien agregar un latch pero esto afectará al sincronismo por el hecho de agregar un ciclo de clock. Se hizo el análisis con diferentes frecuencias en el clock wizard y se dio con que la frecuencia **85 MHz** es la que el valor ya es negativo y se tiene que tomar medidas si es que necesita usar esa frecuencia.

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	-0,298 ns	Worst Hold Slack (WHS):	0,069 ns
Total Negative Slack (TNS):	-0,429 ns	Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	11	Number of Failing Endpoints:	0
Total Number of Endpoints:	2517	Total Number of Endpoints:	2517

Timing constraints are not met.

Intra-Clock Paths - clk_out1_clk_wiz_0 - Setup

Name	Slack ^1	Levels	High Fanout	From
Path 1	-0.298	11	76	PIPELINE/EX_MEM_reg/selected_reg_reg[2]/C
Path 2	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 3	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 4	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 5	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 6	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 7	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 8	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 9	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C
Path 10	-0.013	7	14	UART_debug_unit/baudrate_gen/counter_reg[5]/C

7 - Conclusión:

Este trabajo se basó en la creación de un procesador MIPS de 32 bits con una arquitectura de 5 etapas se logró contemplar riesgos estructurales y de datos.

También se agregó una unidad de depuración con interfaz de usuario para la gestión y análisis desde una PC a través de comunicación UART.

El análisis de timing revela un rendimiento sólido, alcanzando una frecuencia máxima de 85 MHz. Estos resultados respaldan la fiabilidad y eficiencia del diseño, destacando su idoneidad para aplicaciones de alto rendimiento.